THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 58, No. 6, July-August 1979
Printed in U.S.A.

Traffic Service Position System No. 1:

Software Development Tools

By J. J. STANAWAY, Jr., J. J. VICTOR, and R. J. WELSCH

(Manuscript received December 28, 1978)

This paper is concerned with the development tools, strategies, and methodologies employed by Traffic Service Position System (TSPS) application programmers for the creation and testing of TSPS software. Two environments are described: (i) The support environment provided by an IBM 370/168 and related TSPS support software packages for the production of testable TSPS software. (ii) The support environment provided by the TSPS system laboratory, utility system, and specialized hardware for testing the TSPS software.

I. INTRODUCTION

This paper describes the tools of TSPS software generation and testing in the following environments:

(*i*) The facilities provided via a general-purpose IBM processor for the production of testable TSPS software.

(*ii*) The facilities provided by the TSPS system laboratory for testing new and changed TSPS software in an operational environment.

Section II is concerned with the IBM support environment utilized by TSPS application programmers. The major tools and strategies are described. More important, the methodologies that make use of these tools are explained. Section III is concerned with the testing environment provided by the TSPS system laboratory. This environment combines an actual TSPS machine, a utility system and related software, and special hardware for debugging TSPS software.

II. SOURCE CODE DEVELOPMENT AND GENERATION OF SPC-LOADABLE OBJECT CODE

This section concerns itself with the general environment of TSPS source code creation, modification, and preparation for testing using the TSPS system laboratory and associated utility systems.

2.1 General software support environment of TSPS

The software required to operate a particular TSPS installation is comprised of approximately 300 programs (PIDENTs) totaling over 200 thousand 40-bit Stored Program Control (SPC) 1A machine instructions. The combination of these 300 PIDENTs into an issuable (via Western Electric) software package is referred to as a generic release. There are currently four active TSPS generic releases, each having implemented a major new TSPS feature.

The implementation of new minor enhancements are normally provided by a new release of an existing active generic. All capabilities provided by the software of a lower numbered generic are also provided by the higher numbered generic in addition to the new major feature. The starting point for software development of a new generic will be the current state of the source modules comprising the previous generic. Many of these modules remain unchanged in the new generic. Others are modified to produce a new version of the PIDENT that adds the new capabilities for the new generic. In addition, new PIDENTs are created for the new generic. It is therefore possible to have to maintain as many versions of a PIDENT as there are active generics.

2.1.1 Featuring and a single source environment

Four active generics with 300 PIDENTs per generic could imply that 1200 PIDENT source modules would have to be maintained. This is not the case. TSPS employs the use of "featuring" to maintain a single source module for a PIDENT regardless of the number of distinct versions of that PIDENT. Featuring basically means the following:

> Any addition to a PIDENT source module must be bracketed by feature control directives which, during the assembly of the source module, direct the assembler to either assemble or ignore the bracketed source code. Any existing source lines to be replaced/deleted are likewise bracketed.

This implies that a single source for a PIDENT can be maintained which is capable of generating multiple versions of the PIDENT's object module (i.e., the output of the assembler). By appropriate feature control directives to the assembler, different versions of a PIDENT can be assembled. In speaking of multiple versions of a PIDENT due to four active generics, what actually is meant is that multiple versions of a PIDENT's object module are producible from a single PIDENT source module.

The feature control directives employed by TSPS are:

INFOR feature expression

OUTFOR feature expression
ENDFOR feature expression.

The term "feature expression" in all three is a Boolean expression which is evaluated as "true" or "false" by the assembler. INFOR implies "assemble" the following source lines if the feature expression is true. OUTFOR implies "ignore" the following source lines if the feature expression is true. ENDFOR is the terminating bracket for source lines to be assembled/ignored. INFOR with a true feature expression is the same as OUTFOR with a false feature expression. "Feature expressions" may be combined with Boolean "and," "or," and "not" operations.

Each generic has associated with it sets of feature expressions that always evaluate as true. Every assembly of a PIDENT source module is initiated by the processing of a special control statement which specifies the generic for which the PIDENT is being assembled and therefore the feature expressions which are to be "true" for this particular assembly.

2.1.2 Software development support environment

Software to be executed on the TSPS SPC 1A is generated by means of the computing facilities of an IBM processor located at the Columbus, Ohio branch laboratory of Bell Laboratories. This facility operates the OS/370 operating system with the IBM Time Sharing Option (TSO). The latter point is significant because the TSPS development organization resides at the Indian Hill laboratory in Naperville, Illinois. All accesses to the TSPS software data base are through TSO via dial-up terminal access from Indian Hill. The high-speed data network (VIPERDAE) connecting Columbus and Indian Hill allows hard copy and tape output to be returned to Indian Hill.

TSO provides the needed interactive facilities to both TSPS application programmers and TSPS program administration personnel for data base administration, PIDENT creation and modification, and submission of OS/370 batch jobs for assemblies, loads, etc. It should be mentioned that this interactive environment was new for TSPS with the development of Generic 8. Before Generic 8, the software development environment was punched-card-oriented, with all functions to be performed being initiated via over-the-counter submission of card decks.

2.1.2.1 Creation and modification of PIDENT source modules. The major TSO-provided tool utilized by TSPS application programmers for the creation or modification of PIDENT source is the QED text editor. QED is a powerful, flexible, and general-purpose text editing facility capable of either line number or context editing on a range of various OS/370 file organization types. It should be mentioned at this time that TSPS application programmers do not directly modify existing

PIDENT source modules. Since a single-source module is used to generate (via assembly) the object modules for more than one active generic, it is felt that direct modification of the PIDENT source for any particular generic is too dangerous. Instead, TSPS employs the use of two editors for PIDENT source modification. Via QED, the application programmer is actually creating the editor statements to be processed by the Advanced Processor Editor (APE). APE is a very simple, line-number-oriented editor. It provides only the basic "insert," "replace," and "modify" functions and is specifically designed to operate in conjunction with the TSPS assembler. In almost all cases, APE and the TSPS assembler are executed in sequence in the batch environment of the OS/370. APE applies the edits created via QED to the official PIDENT source and outputs a temporary edited copy of the PIDENT source, which is then assembled. The actual PIDENT source module is not altered during this sequence, although the mechanism does exist in APE to permanently apply the edits to the PIDENT source module, renumber all lines sequentially, and regenerate the PIDENT source module. From this point on, any reference to a PIDENT source module is actually a reference to a PIDENT source module in combination with the official module of APE edits for that PIDENT.

2.1.2.2 The TSPS assembler. The creation/modification of TSPS PIDENT source (source + APE edits) is only the first step in a sequence. This sequence of functions will eventually produce an output from the IBM support machine which is capable of being executed and tested on the TSPS system.

The second step in this sequence is the conversion of the PIDENT source module into an assembled object module suitable for input to the load step. This conversion process combines the execution of the APE editor to produce a temporary modified source module, followed by the assembly of this modified source module by the SPC-SWAP (*Swi*tching *A*ssembly *P*rogram) assembler. Primary outputs of the SPC-SWAP assembler are an object module and an assembly listing corresponding to a particular version of the PIDENT. SPC-SWAP is an excellent, high-powered assembler which possesses an assortment of pseudo-operations for controlling listing format, symbol definitions, etc. SPC-SWAP also includes powerful MACRO definition and usage facilities.

Another facility of the SPC-SWAP assembler which is heavily used by TSPS in its multigeneric environment is the capability to create a special file (referred to as a library) of symbol or macro definitions which can then be accessed by subsequent PIDENT assemblies for the purpose of symbol or macro resolution. It is the library and macro facilities of SPC-SWAP which allow the single-source, multiple-generic environment of TSPS to be viable. The feature control directives described in Section 2.1.1 are actually macros available through the

library facility to each TSPS PIDENT assembly. The control statement which initiates each PIDENT assembly is again a macro (the PACKAGE macro). Execution of the PACKAGE macro, which basically takes a generic name as a parameter, establishes the generic environment of the assembly (i.e., what feature expressions evaluate as "true"; what generic-dependent libraries of symbol, macro, and data definitions will be available during the assembly; what "name" should be given to the produced object module; etc.). Simply by "inserting" via APE a different specification for the PACKAGE macro and reassembling, a single-source module is capable of producing many different generic-dependent versions of its object module.

2.1.2.3 The TSPS loading process. Object modules produced by the SPC-SWAP assembler are not suitable as input to the TSPS. The final step in the sequence of operations which produces SPC-compatible output is the execution of the SPC loader on the IBM support machine. Primary inputs to the SPC loader are the object modules for all PIDENTs comprising a particular generic software release and control directives specifying such things as (*i*) what areas of SPC memory are available for loading PIDENTs, (*ii*) what PIDENTs are to be loaded and, if necessary, at what addresses, and (*iii*) what types of maps and cross references are to be generated. In these respects, the SPC loader is very similar to most relocatable linking loaders. Primary output of the SPC loader is an 800-bits-per-inch, 9-track tape representing the relocated, fully linked generic load module which is capable of being read into SPC memory and executed. Corresponding to the tape output is a printed load map showing memory assignments, available space, etc., and optionally cross-reference listings showing entry point definition and PIDENTs referencing.

An additional capability of the SPC loader is somewhat unique to ESS-type loaders and is heavily used during TSPS software development. When creating a full generic software load of all 300 or so PIDENTs, the SPC loader can be directed to create a special file, called a HISTORY, into which detailed information concerning the generated load is written. The information includes the names of all PIDENTs loaded; the addresses at which they were loaded; how much space each consumed; what entry points each defined; where each entry point was referenced; where and how much free SPC memory remains; what SPC memory was originally available to be loaded; and a complete copy of the relocated, linked, and loaded SPC memory. On a subsequent execution of the SPC loader, the HISTORY file can be reinput to provide the capability for what is known as a partial load. On a partial load, the SPC loader need only be informed of changes to be made to the previous full load represented by the HISTORY. Previously loaded PIDENTs can be unloaded or replaced by new versions, new PIDENTs can be added, additional SPC memory can be made available for loading into, and a

new HISTORY reflecting all changes can be generated. Of particular importance is that, on a partial load, the tape that is generated reflects only the difference between the updated load image and the previous load image that had been saved on the input HISTORY file. This partial load facility of the SPC loader provides SPC application programmers with an incremental load capability that is the basis for one of TSPS's primary software development methodologies to be described.

2.1.2.4 The Interactive Program Administration System. IPAS (Interactive Program Administration System) is a tool developed for use by TSPS and other SPC 1A based systems. IPAS executes in the interactive TSO environment and primarily serves to shield the application programmers and program administration personnel from the complexities of the OS/370 operating system, specifically the nontrivial Job Control Language (JCL) required for batch execution of SPC-related support tools. IPAS is based on the concept of PIDENTs and versions of PIDENTs and utilizes the Bell Laboratories Data Management System (DMS), a hierarchical data base system. IPAS provides the TSPS user with access to QED for line edit file creation and a simple command language for initiating the execution of the SPC-SWAP assembler, the SPC loader, and other minor support tools. IPAS was developed for use with all TSPS generics but to date has been most extensively used on the Generic 8 development.

2.2 TSPS software generation methodologies

The discussion to this point has centered on the support tools and basic implementation strategies available for generating and preparing for execution the TSPS PIDENT source modules. The discussion now turns to the methodologies which make use of these tools and strategies. Two methodologies will be covered, one which applies to software generation in a relatively free administrative atmosphere, and another which applies to software generation in a very tightly controlled change environment. Both have been applicable to the recent Generic 8 development, and in fact both were formulated for the Generic 8 development. Both are equally applicable to the other generics. The two methodologies correspond to the two administrative modes under which TSPS application programmers work. The "development" mode implies the free atmosphere; the "frozen" mode, the tightly controlled atmosphere.

2.2.1 Development mode methodology

The "development" mode primarily applies to the generation of a major new feature, and therefore to a new generic. In this mode, the object is to provide the application programmers with as much freedom as possible in generating the new feature. For this reason, there is little restriction on how the programmers modify existing PIDENTs or create

new PIDENTs. The methodology formulated for this "development" environment is heavily based on the team programming concept and the partial load capabilities of the SPC loader.

The overall software development of the Automated Coin Toll Service (ACTS) feature was divided basically along functional lines. TSPS application programmers were organized in programming teams based on these functions. The implementation of a given function required the modification of some subset of the 300 or so TSPS PIDENTs and the creation of new PIDENTs. The functional subsets of PIDENTs were not necessarily mutually exclusive. Many times multiple functions required modification to the same PIDENT. The startup point for the software development of Generic 8 was the stable state of the TSPS software as it existed for Generic 7 in the second quarter of 1976. TSPS program administration personnel set up the proper PACKAGE macro for Generic 8 and established the Generic 8 dependent macro and symbol libraries which the PACKAGE macro would make available to the SPC-SWAP assembler when performing Generic 8 assemblies. All Generic 7 PIDENTs were then reassembled to produce relocatable Generic 8 versions of the PIDENT object modules. Recall that a PIDENT source is actually the combination of the official (i.e., Program Administration Group [PAG] controlled) PIDENT source module and the official file of APE line edits for that PIDENT. All relocatable Generic 8 object modules were then input to the SPC loader to produce a full Generic 8 load module. This load module was designated the "Base 0" Generic 8 load, capability-wise identical to the Generic 7 state from which it was generated, and loadable and executable in the TSPS system laboratory. The foundation upon which to build Generic 8 was established.

The programming teams were now capable of incrementally modifying this "Base 0" load and testing their function implementation. To illustrate the methodology, assume programming teams 1 and 2. The function of team 1 requires modification of PIDENTs A, B, and C, and the creation of a new PIDENT D. The function of team 2 requires modification of PIDENTs A, E, F, and G. Team 1 would proceed as follows:

(*i*) Exact copies of the official line edit files for PIDENTs A, B, and C would be created using QED. Each would be modified as needed for team 1 function implementation.

(*ii*) The source for new PIDENT D is created using QED.

(*iii*) An SPC-SWAP assembly is initiated via IPAS or other TSO facilities utilizing the copied and modified team 1 line edit files for PIDENTs A, B, and C and the created source file for PIDENT D. The object modules produced by SPC-SWAP are saved as team 1 object modules.

(*iv*) Using the SPC loader, initiated via IPAS, a partial load is gen-

erated based on the HISTORY file corresponding to the "Base 0" Generic 8 load. PIDENTS A, B, and C are replaced by team 1 versions, and PIDENT D is added. The partial load tape produced reflects only the differences between the team 1 load and the "Base 0" load.

(*v*) Team 1 is now capable of overlaying their partial load image on top of a known-to-be-stable "Base 0" load image in the TSPS system laboratory and testing their function unencumbered by new code from other teams.

(*vi*) The cycle can be reiterated as problems are discovered during testing, except that modifications are made to team 1 line edit files.

Team 2 has concurrently been developing their function following the same procedures as outlined for team 1. This approach allows very extensive testing of individual functions, comprising large amounts of new and modified software, to be accomplished prior to a large-scale integration of functions.

The PAG personnel again became involved with the Generic 8 development at periodic intervals (usually 6 to 8 weeks) to produce an updated base load. In preparation for performing the official reassemblies for all PIDENTS modified by the programming teams, a merging of official and team line edit files must take place.

Recalling that both teams 1 and 2 copied the official line edit file for PIDENT A, the PAG personnel would first merge the team line edit files for PIDENT A, and the result would then be merged with the official line edit file for PIDENT A to produce an updated official line edit file. PIDENTS modified by a single team required only a single merge. The final merge with the official line edit file guaranteed that no original official line edits had been inadvertently deleted or modified in such a way as to adversely affect generics. Having completed the merging process, PAG could then make any required modifications to Generic 8 macro or symbol libraries, and then reassemble all modified PIDENTS to produce updated official Generic 8 object modules. These object modules now would contain all the function code tested by the individual teams up to the time the new base was created. PAG then reexecutes the SPC loader to produce a "base $n + 1$" load module and corresponding HISTORY file. The base load image in the TSPS system laboratory is then updated to "base $n + 1$," and the new base can be system-tested to ensure stability.

Although a large amount of new and modified software is introduced with a new base load, the interval between the start of the merging process and the completion of the system testing of a new base averages about 2 weeks. During this interval, the programming teams are able to continue working against "base n," with the stipulation that any additional team line-edit changes must be incorporated with

the updated official line-edit files reflecting "base $n + 1$." Due to heavy testing of individual team functions, a minimum amount of system testing is required to stabilize the new base load, even though the functions had not previously been integrated.

Prior to Generic 8, the "development" mode methodology was based on manually overwriting a load image in the TSPS system laboratory to test new functions or including untested software in new load images. System testing of new load images was an enormous, time-consuming task. Function testing via manual overwrites was more of a hindrance to software development than a help.

2.2.2 Frozen mode methodology

The "frozen" mode of software generation applies primarily to active generics which have been issued through the Western Electric Company, and to the latter stages of the development of a new generic. The objective of the "frozen" mode is to maintain the maximum amount of stability in the generic software by providing a highly controlled and documented change environment through which application programmers must make software corrections.

For a generic in the "frozen" mode, software change is initiated in response to a written Trouble Report (TR) documenting a suspected problem or a needed improvement. The "frozen" mode imposes a number of restrictions on the application programmers and PAG as to the manner of software change:

(*i*) No change to a PIDENT can cause the size of the PIDENT's object module to either increase or decrease.

(*ii*) The primary method for testing is not the partial load, but incrementally applied changes to the frozen generic's load image in the TSPS system laboratory.

(*iii*) Line edit changes corresponding to a laboratory change must produce a bit-for-bit match between the PAG-generated (via PIDENT reassembly) next release of the frozen generic and the overwritten old release in the TSPS system laboratory.

(*iv*) All programmer-generated changes must be independently tested and approved by a generic test team.

(*v*) The application programmer must generate a written Correction Report (CR) documenting any changes made in response to a TR.

(*vi*) Not only must the change be independently tested, but the TR, CR, overwrite, and line edits must be approved by a generic software change review committee prior to the line edits being included in the next release of the frozen generic.

To alleviate some of the problems associated with (*i*), (*ii*), and (*iii*), above, a functionally identical set of "patching" directives exists,

available to both the SPC-SWAP assembler and the test laboratory utility system overwrite assembler. In the case of SPC-SWAP, these are TSPS system macros available to any PIDENT assembly. They are merely control directives to the utility system overwrite assembler. Tbese "patching" directives allow the application programmers to add new software, replace existing software, or delete existing software within a PIDENT without altering the assembled size of the PIDENT.

Primary input to utility system overwrite assembler is a deck of punched cards, composed of symbolic SPC source to be assembled, and utility system control directives. A few major restrictions on the change to be assembled are:

(*i*) There is no macro capability. Any macro to be assembled must be manually expanded prior to input.

(*ii*) There is no LIBRARY facility as with the SPC-SWAP assembler. The utility system overwrite assembler must be explicitly informed of the value of any and all symbols referenced by the overwrite but not defined within the overwrite.

(*iii*) No arithmetic beyond addition and subtraction is allowed.

(*iv*) Many data defining pseudo-ops of the SPC-SWAP assembler are not recognized by the overwrite assembler.

Given these restrictions of the overwrite assembler and the previously stated restrictions of the "frozen" mode environment, the change implementation flow prior to Generic 8 (and therefore prior to general TSO usage by TSPS) went as follows:

(*i*) The TR would be received by the application programmer who would be making the software change.

(*ii*) The application programmer would generate an overwrite deck to fix the stated problem and test the fix by temporarily overwriting the generic load image in the system laboratory.

(*iii*) Satisfied with the results, the application programmer would generate the corresponding CR and a deck of line edits for the PIDENT source which produced results identical to the overwrite.

(*iv*) The TR, CR, overwrite deck, and line edit deck would then be submitted to the generic software change review committee for approval. If rejected, back to step (*ii*).

(*v*) If approved, the TR, CR, overwrite deck, and line edit deck are submitted to the generic system test team for independent overwrite testing. If rejected, back to step (*ii*).

(*vi*) If approved, the TR, CR, overwrite deck, and line edit deck are submitted to PAG. The line edit deck is included in the official line edit deck for the PIDENT(s) involved, the TR and CR are filed, and the overwrite deck is saved.

This procedure was followed for each TR requiring a software change. When a new release of the frozen generic was required, PAG would

reassemble all modified PIDENTs and regenerate the generic load, maintaining the starting address and size of each PIDENT. This new load was required to exactly match the overwritten load image in the TSPS system laboratory before it would be released to Western Electric for distribution. The methodology outlined above was successful but possessed inherent and painful shortcomings when it was time to generate and match the next release of the frozen generic. The shortcomings were primarily due to the overwrite assembler and the need to create a separate line edit deck producing identical results. The areas most susceptible to error were the manual symbol resolution and manual expansion of macros required by the overwrite assembler, but not required in the line edits.

For Generic 8, the basic theory of this overwrite methodology, with its checks and approvals, was not substantially altered. There was, however, the development of a new tool, an Overwrite Generation (OVGEN) program, which eliminated the need for separate manual generation of overwrite and line edit decks. OVGEN allowed application programmers to continue to create line edits as files via QED. OVGEN requires the application programmer to include in the line edit file special control directives, identified by the programmer I.D. and a trouble report number, which informed OVGEN of which line edits to extract for overwrite generation. OVGEN, actually a combination of three pre-processors, an SPC-SWAP assembly, and a post-processor, outputs a symbolic overwrite deck for input to the utility system overwrite assembler. The overwrite deck contains all information needed for external symbol resolution. Due to the fact that SPC-SWAP is used to assemble the line edits, the application programmer can utilize macros, libraries for symbol resolution, the SWAP data defining pseudo-ops, etc. In other words, for the application programmer, the environment is quite similar to the partial load environment, with final output being an overwrite deck instead of a partial load tape.

The advantages of the OVGEN procedures in the frozen mode are many.

(*i*) The application programmer need only create the line edits in response to a TR.

(*ii*) The macro facilities of SPC-SWAP are available for use.

(*iii*) Symbols used which are external to the line edit are resolved via pre-processing and SWAP LIBRARY facilities.

(*iv*) Line edits are individually assembled. This leads to far less assembly problems by PAG when the full reassembly of the PIDENT is performed for the next release of the frozen generic.

(*v*) The final match between the new release of the frozen generic and the old release plus overwrites is considerably cleaner due to the overwrites having been generated directly from the line edits.

III. LABORATORY TESTING ENVIRONMENT

TSPS programmers use the TSPS system laboratory complex to test and debug new or changed PIDENTs. One or more programmers working on similar program areas will schedule time in the lab. Thus, this complex has been designed to provide a working environment conducive to high programmer productivity.

Two TSPS laboratories are available to programmers. Both consist of the Stored Program Control (SPC) No. 1A/TSPS complex, the nonresident utility system, and call-oriented simulators. The SPC/TSPS complex allows programmers to test in an environment much like a typical TSPS office. The utility system and simulators provide debugging aids not found in a typical office. This section describes the hardware and software facilities which make up the laboratory complex.

3.1 Hardware configuration

The TSPS laboratory configuration is shown in Fig. 1. This section discusses each component in the configuration except the simulators, which are discussed in Section 3.3.

3.1.1 Stored program control (SPC) 1A

The TSPS is controlled by the SPC 1A.[1] The SPC consists of a processor, memory system, central pulse distributor, signal distributor, master scanner, and a maintenance control center for the man-machine interface. Each of these parts is duplicated for reliability except the maintenance control center.

The SPC processor provides the control for the TSPS by executing the instructions in the memory. The processor cycle is 6.3 μs. The registers available in the processor include a 20-bit Program Address Register (PAR), 20-bit Address Image Register (AIR) which contains the address of the most recent store access, 47-bit Memory Access Register (MAR) for storing the information read from or written into the memory, and seven 20-bit index registers. The index registers are general purpose and may be used for any function.

The word length of the SPC memory is 47 bits (40 bits of information and 7 for error correction). There are 20 bits of addressing. Nineteen bits select the memory word. The remaining bit determines which half of the word is used.

The processor communicates to the peripherals and TSPS by three units: the Central Pulse Distributor (CPD) which allows the SPC to send pulses to points in the system where fast response to an instruction is needed, the signal distributor which allows the SPC to operate or release magnetic latching relays which are connected to output points, and the master scanner which provides status and supervisory inputs to the SPC from the various units in the SPC complex.

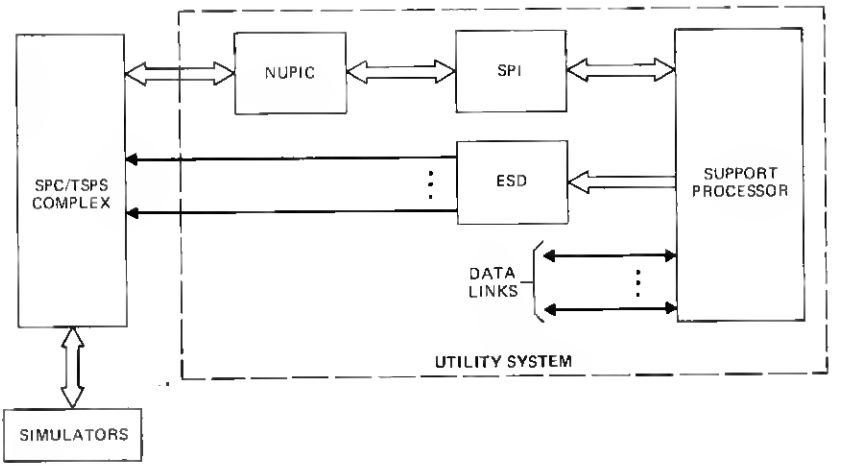

Fig. 1—TSPS system laboratory.

The Maintenance Control Center (MCC) consists of a control and display panel, a teletypewriter, and a program tape unit for loading the SPC 1A memory.

3.1.2 Nonresident utility system

The utility system provides a means for easily loading and reading the SPC memory, for debugging real-time programs in a noninterfering fashion, for controlling devices in the laboratory, and for transmitting or storing files. The advantage of this utility system over the earlier TSPS utilities is that the programs are nonresident to the SPC and that debugging of programs can be done without interfering with the execution of the system's real-time programs.

The utility system is a combination of hardware and software. This section discusses the hardware aspects of the system, and other sections discuss how the utility system is used for software change administration and program testing. The utility hardware consists of a support processor, a support processor interface, a noninteracting utility program interface console, an electronic signal distributor, and serial data links.

3.1.2.1 Support processor. The support processor used in the utility system is a commercial minicomputer. The minicomputer uses the XVMDOS operating system and its peripherals include a 9-track magnetic tape unit, fixed head disk, 3-disk-pack bulk memories, teletypewriter, high-speed printer, card reader, and paper tape reader and punch.

User files and TSPS generic programs are stored on the disk memory. These files can be updated and additional files added by means of the

magnetic tape unit, card reader, or paper tape reader. The user input is via the teletypewriter and card reader. The output is usually over the high-speed printer.

3.1.2.2 Noninterfering utility program interface console/support processor interface (NUPIC/SPI). The interface between the SPC and the support processor is the Noninterfering Utility Program Interface Console (NUPIC) and the Support Processor Interface (SPI). The NUPIC and SPI are two separate circuits but are discussed together because they are so closely related.

The NUPIC interfaces directly with the SPC processors to allow the user to monitor and control the system programs. It provides access to the PAR, AIR, MAR, and the index registers in each SPC processor. It also provides processor clock control and interrupt interfaces. The circuitry in the NUPIC allows the user to load the SPC, read the SPC memory, set program matchers, and have the contents of the SPC registers read and stored. The NUPIC has a man-machine interface consisting of SPC register displays and manual controls. These manual controls are particularly useful if the support processor should fail. The section on program testing will discuss the debugging aids available with the NUPIC.

The NUPIC is controlled by the support processor via the SPI circuit, which provides a 2-way, high-speed data communications channel. Since the support processor and the SPC processors have different word lengths and different cycle times, buffering (core memory) is provided in the SPI. The SPI is used in conjunction with the NUPIC and the support processor for loading the SPC memory, reading the SPC memory, and collecting data during program debugging.

3.1.2.3 Electronic signal distributor (ESD). In testing programs, it is often necessary to have TSPS configured differently due to multiple generics. This means that specified circuits can be connected to or removed from the TSPS buses, equipment can be removed from service, or equipment can be put into service. The ESD provides a means to do this automatically. Up to 2048 distributor points can be individually set or reset by the support processor. These points can be used to control relays, lamps, and logic inputs.

Each user can have a file on the support processor which specifies the generic program and how the ESD points should be set. Section 3.2.1.2 discusses the creation and execution of the user files.

Another use of the ESD is for physical fault insertion. This application is discussed in a later section on trouble location manual generation.

3.1.2.4 Serial data channels. Several serial data channels are available on the support processor. The channels are full duplex, with data rates up to 10K baud. The channels are used for transmitting

files or receiving files for storage. Presently, one of these channels is used for transferring user call load files to and from the microprocessor controlled call simulator (Section 3.3.3.2) and another is used for loading the writable storage unit (used in program development) of the Programmable Controller in the Station Signaling and Announcement Subsystem for the Automated Coin Toll Service[2] (ACTS) feature.

3.2 Laboratory software change administration

The desire to add new features, as well as the need to correct software errors, make it necessary to be able to easily change programs in the TSPS laboratory. This is done differently, depending on the state of the software base being changed. If the code is being developed into a new major generic issue, programmers are free to perform large-scale code modifications and additions. This is termed the development mode. If the base is already an official generic issue, only minor changes and additions can be made in response to specific troubles. This is done to minimize the need for extensive retesting after applying the change. This mode is termed the frozen mode.

This section deals with the tools available for changing the TSPS program in the lab in both of the above modes.

3.2.1 Development mode

As mentioned in Section 2.2.1, the partial load is the primary means of changing code in the development mode. A partial load tape containing all new and changed object code is brought into the TSPS lab by the programmer for debugging. The programmer then uses two support processor programs to load the base load plus the partial load into the SPC. These programs are DZLOAD and ALCFG (Automatic Laboratory Configuration). Once the desired load is in the SPC, the NOVA (Noninteracting Overwrite Assembler) program is used to make minor code revisions until a new partial load tape can be generated.

3.2.1.1 The DZLOAD program. DZLOAD is the interchange and comparator program for SPC code and data residing on magnetic tape, disk, or SPC memory. It allows the user to easily load and verify SPC programs as well as create duplicate copies of SPC memory on disk or tape. The user may choose to store a partial load tape on disk in the support processor, which eliminates the need to carry the magnetic tape into the lab for subsequent debugging sessions.

DZLOAD deals with only one file at a time; however, the support processor can also load multiple files and control the TSPS lab's hardware configuration. The program that does this is called ALCFG.

3.2.1.2 The ALCFG program. The TSPS system laboratories are used to develop and test hardware and software for use at TSPS installations in the field. The laboratory is used to simulate configu-

rations existing in the field and new hardware configurations under development.

The ALCFG program allows the user to easily and quickly change the laboratory's program and hardware configuration. The result is increased lab availability, brought about by a decrease in the manual action required to establish a particular configuration. The user can predefine a configuration and store it on a support processor disk. Thus, calling in the configuration definition under ALCFG will cause the quick reestablishment of the desired lab environment—both software and hardware. The lab hardware is controlled by the Electronic Signal Distributor (ESD). The ESD is explained in Section 3.1.2.3. ALCFG allows the user to easily define a particular ESD state.

During a lab testing and debugging session, the user will probably uncover minor coding errors or oversights. These errors can be temporarily corrected in SPC memory using the NOVA program.

3.2.1.3 The NOVA program. NOVA is a utility program that allows the lab user to overwrite SPC memory. The input to NOVA is symbolic SPC source code residing on punched cards or disk, or typed directly on the user terminal. Since the program being changed resides at a fixed SPC memory location and occupies a fixed amount of storage space, code additions must be incorporated in a "patched" fashion. NOVA is therefore designed to manage a series of patch buffers. These buffers are the actual start and end addresses of spare program memory in the SPC. Definitions of these buffers are covered in Section 3.2.2.2.

NOVA allows the user to specify program changes in either relocatable or absolute fashion. NOVA reads the overwrite statements, assembles them into SPC object code, prints a listing, and stores a binary image of this code on a support processor disk. This binary file is then loaded into SPC memory. Another binary file is also kept by NOVA, namely, the contents of the addresses specified in the overwrite prior to loading the overwrite. This file is identified by a temporary overwrite number. The user can therefore instruct NOVA to flush a specific overwrite out of SPC memory by restoring all addresses to their contents prior to the change.

Many options are available to the NOVA user. Some commonly used ones are (*i*) assemble and produce a listing, (*ii*) assemble, produce a listing, and create a binary file, (*iii*) load the last binary file created, and (*iv*) print the old data along with the overwrite listing. Another option has to do with permanent overwrites. This is covered in the next section.

3.2.2 Frozen mode

The NOVA-assembled overwrite is the primary means of changing SPC code in the frozen mode (see Section 2.2.2). The main emphasis in

this mode is placed on incremental change documentation and testing, and administration of the updated generic base load. The NOVA program is used extensively during this process. A programmer submits, in addition to TR/CR (trouble report/correction report) forms and line edits, a symbolic overwrite to a system test team. This overwrite is designed to fix a particular trouble existing in the base generic. The test team tests the change as a temporary NOVA overwrite and, if accepted, prepares to permanently incorporate it in the base generic.

3.2.2.1 NOVA—Permanent overwrites. The mechanics involved in establishing a permanent NOVA overwrite are very similar to those for temporary overwrites. The main differences are

1. No old data file is kept (permanent overwrites cannot be flushed from SPC memory).
2. The pointers into the patch buffers are permanently changed to indicate that patch used in the overwrite is no longer spare program memory.

In addition, a permanent record of the start and end address of individual patches is kept on disk. This information is extremely useful to PAG when generating a patched load (see Section 2.2.2) corresponding to the updated lab base load. Each patch origin must be defined to allow the SWAP assembler to correctly assemble the patched code into the changed program. The NOVA permanent overwrite listing is used to document the change in the lab until the new PAG load and listings are produced.

3.2.2.2 NOVA—Generic administration. Up to this point in the lab software change section, only one base load is mentioned. However, as stated in Section 2.1, more than one TSPS generic issue is usually active at one time. Consequently, NOVA must be able to properly change and patch programs for each active generic. This generic is identified by the user each time the NOVA program gets called in. NOVA uses this identification to access a set of PIDENT and PATCH files unique to that generic. The administration of these PIDENT and PATCH files is handled by a portion of NOVA, called PIDAM. The system test group usually takes care of this administration. PIDAM allows the user to define and update the PIDENT and patch files for each generic. These files contain a list of all PIDENTs that make up the generic issue along with their start and end addresses, available patch buffers, a store patch map (i.e., a map linking SPC programs to a specific patch buffer), and a list of all permanently loaded patches (see Section 3.2.2.1).

3.3 Laboratory program testing

After the SPC has been loaded with a new generic program (development mode) or changes made to an existing generic program (frozen mode), the programmer is ready to begin testing. The following sec-

tions describe the hardware, software, and simulators available to the programmers for testing programs.

3.3.1 Hardware for program debugging

The NUPIC discussed in Section 3.1.2 gives the user extensive program debugging tools. These tools include matchers, visual displays, and processor controls (Fig. 2).

The matchers available with the NUPIC include:

Address Matchers—There are nine address matchers (one manual) which can be set to indicate when a specified program address is executed or when a specified data address is read or written.

Range Trap Matchers—Two range trap matchers are available. These matchers will indicate when any address within a range of program is reached or when any address within a range of data is accessed.

Bit Matchers—Two 20-bit matchers allow the programmer to match against an address, contents of a memory location, or the contents of an index register. The bit pattern to be matched against can have an associated mask. This mask will indicate which bits in the pattern are "don't cares."

Peripheral Matcher—The peripheral matcher is used to indicate when a particular peripheral order accesses a particular peripheral unit. The peripheral unit address can have an associated mask.

Each of these matchers can be set manually via keys on the NUPIC. All but the manual matcher can also be set automatically. When a matcher "fires," the contents of the SPC registers can be collected by the support processor, analyzed, and printed on the high-speed printer.

The matchers can be set up with different options which will be executed when the matcher fires. These options can be interfering and noninterfering. The noninterfering options include register snaps and transfer traces. The transfer trace gives a record of every transfer that occurs in the program once the matcher fires.

The interfering options cause an interrupt in the SPC program execution and transfer of control to the SPC resident utilities. The options include doing a write into unprotected memory, dumping portions of memory, writing to registers, jumping to a different address, and stopping the SPC.

The visual displays on the NUPIC include binary lamp displays for the major points in each SPC processor such as the index registers and buses. Octal displays are provided for the registers most often used. These include the PAR, AIR, MAR, selected matcher address, and relocatable address (least five significant digits). The PAR, AIR, and MAR displays are available for each processor.

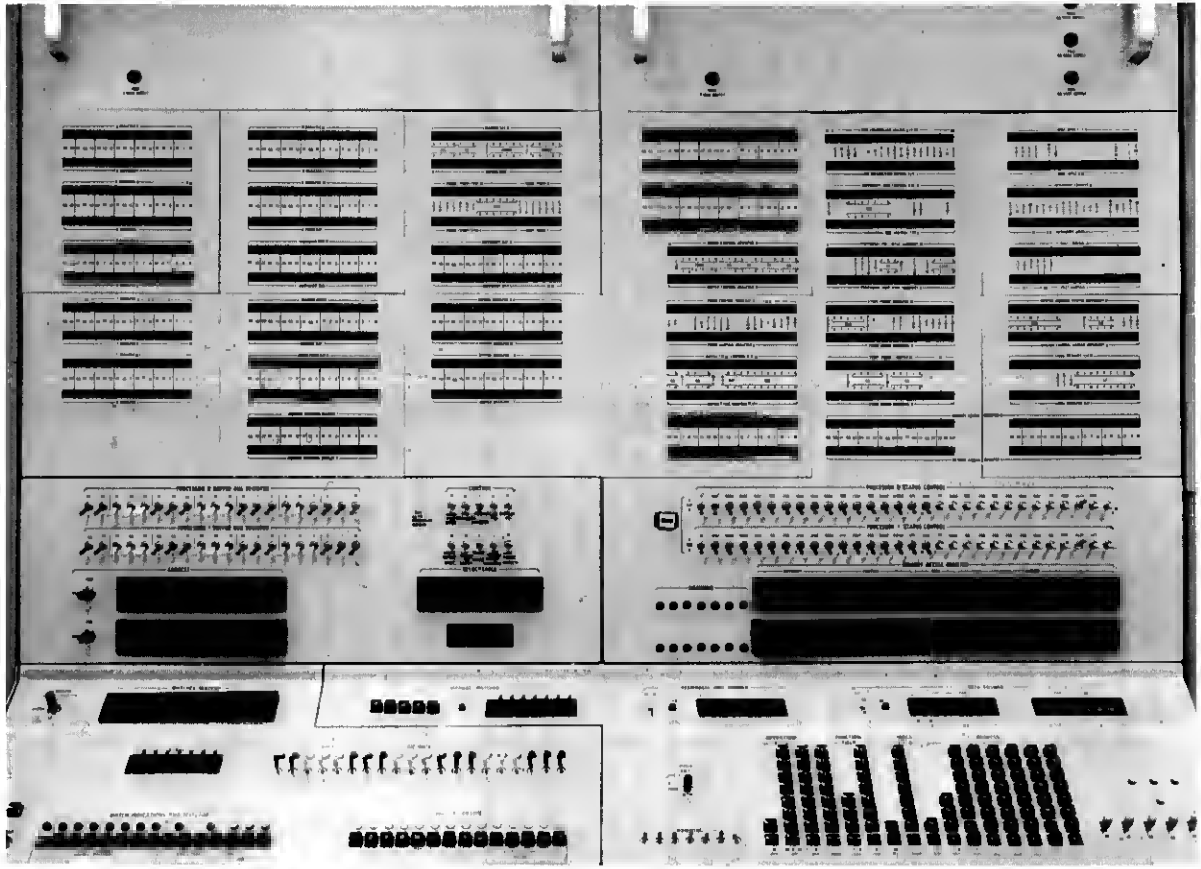

Fig. 2—Noninterfering Utility Program Interface Console.

Several manual control functions are provided to allow the user to selectively control either or both SPC processors.

(*i*) The user can stop or start the processor clock and can step through instructions one cycle or one phase at a time.
(*ii*) Instructions, data, and scanner answers can be inserted.
(*iii*) Matchers can be set up.
(*iv*) Utility interrupts can be generated and flags provided for the SPC resident utility programs.
(*v*) Transient store errors can be simulated.
(*vi*) The processors can be split into two independent systems.

3.3.2 Software for program debugging

Facilities exist in both the support processor and the SPC to aid in TSPS program debugging and testing. The most useful of these resides in the support processor and is called SPCDDT (Stored Program Control Dynamic Debugging Tool). This program activates matchers and traces via the NUPIC to give the user information about program flow in the SPC. Another useful support processor program is AMADMP (Automatic Message Accounting Dump). This program aids call processing programmers by providing formatted AMA information at the completion of a call. Finally, SPC-resident code assembled as a special set of Feature Assembly Debugging Aids (FADA) allows the programmer to selectively control the execution of TSPS programs.

3.3.2.1 The SPCDDT program. SPCDDT is used to set address matchers, range traps, bit matchers, and a peripheral matcher in the NUPIC circuit (see Section 3.1.2). The user has the ability to control certain actions before and after a matcher fires. Some of these options are:

(*i*) Jump to a program address when a matcher fires.
(*ii*) Dump the contents of specific SPC memory areas on the line printer when a matcher fires.
(*iii*) Write information into an SPC memory location when a matcher fires.
(*iv*) Start or end a transfer trace when a matcher fires.
(*v*) Selectively print trace information on the line printer.
(*vi*) Continually store trace information in the core memory of the SPI circuit (see Section 3.1.2.2). If the core fills, overwrite it with the more recent information (overlay mode). Freeze it and send the contents to the support processor for printing when another matcher fires.

Option (*vi*) is a particularly useful feature. It allows the programmer to monitor the entire flow of one (or more) program(s), and print out only the flow prior to an interesting event.

3.3.2.2 The AMADMP program. AMADMP provides formatted AMA

information on the support processor line printer at the completion of a TSPS call. This information is very useful to a programmer in debugging call processing or billing-oriented software.

Normally, the AMA Data Accumulation Program (AMAC) stores the billing information in buffers in TSPS memory. When a buffer is full, AMAC activates another program which records all buffered data on magnetic tape. This tape must then be processed on an off-line computer. The delays and logistics involved in this procedure make it undesirable for debugging purposes. Consequently, AMADMP was written during the development stage of the Remote Trunk Arrangement (RTA) feature of TSPS Generic 7.

AMADMP uses the NUPIC to activate two matchers in the AMAC program. One matcher fires at the start of AMAC's recording of the billing information and the other fires after all information has been buffered for a call. Using the information appearing in the SPC registers at the time the matchers fire, AMADMP requests a dump of the buffer locations used by AMAC. The data from these locations are then formatted in the support processor to make it more readable, and then printed on the line printer.

3.3.2.3 The FADA feature. FADA is a collection of debugging software that can be assembled into a development base load. It consists of special code added to generic TSPS programs as well as a special PIDENT (ECDB). This code does not get released officially for use in live TSPS offices.

FADA was developed during the early stages of Generic 7 as an aid in the recovery of new program loads and a debugging tool. It accomplishes this by allowing the user to selectively inhibit execution of many program functions. This capability makes it possible to simulate many low probability events, cause race conditions, and exercise program failure legs.

3.3.3 Simulators

3.3.3.1 Single-call simulators. Many times in testing programs, the programmer needs the ability to place a single call through the TSPS. Two types of test facilities are available for making single calls. The first of these facilities is the "single-line" simulator. This simulator consists of a local (calling) telephone connected to the local office side of the simulator and the toll (called) phone connected to the toll side of the simulator. The local and toll sides of the simulator are connected to a TSPS incoming trunk. This trunk is dedicated to a particular traffic type. For each traffic type (coin, hotel/motel, RTA), there is a single-line simulator.

Calls are placed by the programmer in the same way a customer would make a particular type of call. The simulator performs all

necessary signaling required by TSPS. The call is recognized by TSPS and handled appropriately by routing it to an operator's position or by connecting the called telephone.

The single-line simulator is used for calls which are to be handled normally. The simulator generates the correct KP, Start (ST) digit, and Automatic Number Identification (ANI) digit for the call. However, there are times when the programmer must have more control over the call. For instance, the programmer may wish to test a call using an improper ST digit or ANI digit. In these cases, the manual trunk test set (also known as the "Burelback Box") is used.

The manual trunk test set allows the programmer to place a single call and to have control over the entire call. The programmer selects the type of TSPS trunk circuit to be used and the call type. By operating switches, the programmer simulates seizure by the local office and responds to supervision from TSPS by keying in the KP digit, the call digits, a ST digit, an ANI digit, and the calling digits. When the toll side of the trunk is seized by TSPS, the programmer generates the toll supervision signals to TSPS. On ACTS calls, the coin tones can also be generated.

3.3.3.2 Multiple-call simulators. There are program bugs which do not show up until there is a substantial traffic load on the system. In TSPS testing, a simulated load can be generated by the Electronic Load Box (ELB) and the Microprocessor Controlled Load Box (MICLOB). Both of these "load boxes" automatically generate calls on multiple TSPS trunk circuits. Each load box simulates the functions of both the local and toll offices.

The characteristics of the ELB are:

(*i*) Generates up to 14 simultaneous calls. All calls must be the same type and the same length.
(*ii*) Can generate 1800 calls per hour.
(*iii*) Works with MF (multifrequency) trunks, 2-wire, or 4-wire, loop or E&M signaling.

To use the ELB, the user selects the number and the type of the calls desired. If more than one call type is required, another ELB must be used. To provide the user with more flexibility in setting up a call load and to provide the capability for coin signaling for ACTS, the MICLOB was developed. MICLOB (Fig. 3) has the following characteristics:

(*i*) It generates up to 32 simultaneous calls of any call mix the user wishes.
(*ii*) The call types include coin, noncoin, hotel/motel, and international.
(*iii*) All call parameters are under user control.
(*iv*) It works with either 2-wire or 4-wire trunks with MF or DP (dial pulsing) signaling and with loop or E&M supervision. Other

Fig. 3—Microprocessor Controlled Load Box.

trunk types can be easily handled by modifying the microcomputer program and providing the proper trunk interface.

(*v*) The call load can be as high as 10,000 calls per hour for short calls.

The user sets up his call load by entering the call parameters for each trunk via a teletypewriter or a terminal. Once the load is established, the calls can be started or stopped individually or as a group. The user can save his call load parameters on the support processor's disk via a serial data link (Section 3.1.2.4). This call load can be reloaded at a later time. This save/load capability allows users to set up a call load without entering the information via the teletypewriter each time.

Another feature of the MICLOB is the error messages printed on the TTY or terminal whenever calls do not proceed properly. These messages can alert the user to a problem with a trunk circuit or a MICLOB circuit.

In a testing environment where a heavy load is required, the ELB can be used to generate a background load of a particular call type. The MICLOB can be used to generate a load of special calls or of call types not possible with the ELB. In this manner, the TSPS programs can be exercised in the lab much like a live system.

3.3.3.3 Operator simulators. Operator simulators are used to simulate operator actions on calls arriving at positions. These simulators are used when a call load is generated containing calls which require operator assistance. The simulators recognize the call types arriving at the position and generate the required keying sequences. There are two types of operator simulators used in TSPS. The older simulator is hardwired. It can handle all call types except ACTS. Each of these simulators requires an actual position to operate.

The new Microprocessor Operator Position Simulator (MOPS) can handle all call types. Each call type can be handled differently, new call types can easily be added, and existing calls can easily be changed. This simulator can work with or without an actual position available. In this way, less positions are required in the laboratory. The advantage of having calls go to an actual position is that one can observe how the simulator is handling calls or one can manually handle a call if necessary. These functions can be duplicated via a terminal connected to the simulator.

The lamp and display orders sent to a simulator position by TSPS during a call can be converted to lamp names and digital displays and printed on the terminal. By monitoring a particular simulator position, the programmer can observe calls going to the position. The keys on the terminal are programmed to act as position keys so that the programmer can also manually handle a call if desired. The terminal can be remoted if necessary. This feature is useful when the TSPS

operator positions are not at the same location where program testing is being performed.

3.4 Trouble location manual (TLM) generation

The Trouble Location Manual (TLM) contains Trouble Location Numbers (TLN) for a circuit or subsystem. Associated with each TLN is the location of probable failing circuit packs in the circuit. These numbers which are an output from the diagnostic program are used to assist craft in fixing a faulty circuit.

The generation of a TLM for each circuit or subsystem requires that faults be physically inserted in these circuits. The diagnostic program for the circuit being faulted is run. The results from the diagnostic are used to generate the TLN. The collection of TLNs and faulted circuit locations make up the TLM.

Originally, in TSPS, most circuit pack faults could be inserted at the connector. However, with the introduction of RTA and the IGFET (Insulated Gate Field Effect Transistor) memory, circuit packs become much more complex. They contained many integrated circuits in Dual In-Line Packages (DIP). All necessary faults could not be inserted at the connector. To insert faults in the newer circuit packs and to speed up the TLM generation process, a minicomputer-controlled fault insertion technique was developed. This technique, which consists of fault insertion hardware and software, is discussed in the following sections.

3.4.1 Physical fault insertion

The type of faults inserted in the circuits are opens and shorts to desired voltage levels on the circuit pack connectors and the pins of the DIPs on the packs.

The circuit pack which is being faulted is a special version of the standard circuit pack in that the integrated circuits are socket-mounted. The circuit pack is mounted in a special extender board which is plugged into the circuit pack connector. The DIPs on the circuit pack have "daughterboards" inserted between them and their sockets. Both the extender board and "daughterboards" have relays which can be controlled to open or short the circuit pack connector pins and the DIP pins.

These relays are controlled by the support processor by setting selected points in the ESD. The fault insertion hardware decodes this information to operate the selected relays and thus insert the desired fault.

3.4.2 Diagnostic control and raw data accumulation

The support processor contains software to control automatic physical fault insertion and the SPC diagnostic programs. It also contains routines to gather the test results, or raw data, from these diagnostics.

These results are combined into a data base used to produce system TLMs. The support processor software designed to control these processes is called MCFIT (Minicomputer Controlled Fault Insertion Technique). The MCFIT program, together with special code in the SPC to interface with system diagnostics, make up the automatic physical fault insertion control software.

3.4.2.1 The MCFIT program. MCFIT was designed to allow rapid fault insertion and TLM generation. The majority of faults to be inserted are stored in a data base on the support processor's disk. This data base consists of all standard faults that are defined for each type of DIP used in the circuit being faulted. Thus, the user must only input the layout of the circuit pack to be faulted and any modifications to the standard fault list. MCFIT automatically controls the faulting of the entire pack, requiring manual intervention only to move the fault insertion hardware to the next pack.

Once the user has specified the pack layout, MCFIT retrieves the correct list of faults from disk and applies any necessary modifications. This list, together with the subsystem identification and circuit pack location, comprise the MCFIT "work file." MCFIT then executes a fault insertion program which sequentially inserts all faults appearing in the work file. This program controls the fault insertion hardware, activates a special SPC program which requests diagnostics on the specified subsystem unit, and records the failure data on the support processor's magnetic tape unit. It also prints summary data on the line printer. These data point out unexpected diagnostic ATPs (All Tests Passed) and inconsistent failure data for the user to examine.

3.4.2.2 SPC interface software. As mentioned above, the MCFIT program activates a special SPC program to request diagnostics. This SPC program is not part of the official generic, but is loaded into memory at the start of each fault insertion session in the lab. The diagnostic is requested through the NUPIC/SPI interface. An interrupt is generated in the SPC which transfers control to this diagnostic interface program. This program sets the appropriate bits in the diagnostic request words and status words corresponding to the subsystem unit being faulted. The diagnostic sequence proceeds normally in the SPC with one exception. Normally, each diagnostic transfers control to the SPC Diagnostic Output Control Program (DOCP) to print the pass/fail data on the maintenance teletypewriter. This special program, however, intercepts the data passed to DOCP and sends them to the support processor via the NUPIC/SPI.

IV. CONCLUSION

Effective software development depends very heavily on adequate development tools and test facilities. The tools and facilities described

in this paper came about through an evolutionary process. They started out in a much more basic form and were improved and expanded many times before they reached their present state. Thus, effective support software and hardware requires ongoing development. This point must be kept in mind so that programmer productivity can continue to improve.

V. ACKNOWLEDGMENTS

The support tools discussed in this paper are the combined effort of many people. Contributions to these tools were made by B. E. Holmes, J. R. Petty, and E. G. Pflaum in the area of program administration, by G. M. Jensen in the utility software, by M. R. Harder, P. L. Shepherd, and G. L. Taylor in the design of the various simulators, and by R. H. Allen and F. H. Ross in maintaining the utility and system laboratory hardware.

REFERENCES

1. G. R. Durney, H. W. Kettler, E. M. Prell, G. Riddell, and W. B. Rohn, "Stored Program Control No. 1A," B.S.T.J., *49*, No. 10 (December 1970).
2. M. Berger, J. C. Dalby, E. M. Prell and V. L. Ransom, "TSPS No. 1: Automated Coin Toll Service: Overall Description and Operational Characteristics," B.S.T.J., this issue, pp. 1207–1223.